SEARCH RESULTS [PDF Full-Text (504 KB)] DOWNLOAD CITATION

Effectiveness of caching policies for a Web server

Narasimha Reddy, A.L.
Texas A&M Univ., College Station, TX;

This paper appears in: **High Performance Computing, 1997. Proceedings Fourth International Conference on**
12/18/1997 -12/21/1997, 18-21 Dec 1997
Location: Bangalore , India
On page(s): 94-99
18-21 Dec 1997
References Cited: 10
Number of Pages: xxiii+539
INSPEC Accession Number: 5767627

Abstract:
We look at a number of policies for managing a file system cache in a Web se Traces from NCSA World Wide Web (WWW) server are used in this study. We several caching policies for improving the overall performance of such a serve show that request response time can be improved by some of the replacemer policies that take size of the request into account. It is shown that least frequ used (LFU) performs well when cache sizes are small (<16 MB) and SpacexAc replacement policy performs consistently better than LRU. We show that prev proposed cache sharing policies improve performance significantly in clustere servers

Index Terms:
Internet cache storage network servers storage management

Documents that cite this document
Select link to view other documents in the database that cite this one.

SEARCH RESULTS [PDF Full-Text (504 KB)] DOWNLOAD CITATION

Home | Log-out | Journals | Conference Proceedings | Standards | Search by Author | Basic Search | Advanced Join IEEE | Web Account | New this week | OPAC Linking Information | Your Feedback | Technical Support | Ema No Robots Please | Release Notes | IEEE Online Publications | Help | FAQ| Terms | Back to Top

Effectiveness of Caching policies for a web server *

A. L. Narasimha Reddy

Texas A & M University
214 Zachry
College Station, TX 77843-3128
reddy@ec.tamu.edu
Phone: 409-845-7598
FAX: 409-845-2630

Abstract

In this paper, we look at a number of policies for managing a file system cache in a web server. Traces from NCSA World-Wide-Web (WWW) server are used in this study. We study several caching policies for improving the overall performance of such a server and show that request response time can be improved by some of the replacement policies that take size of the request into account. It is shown that Least Frequently Used (LFU) performs well when cache sizes are small (< 16MB) and SpacexAge replacement policy performs consistently better than LRU. We show that previously proposed cache sharing policies improve performance significantly in clustered web servers.

1 Introduction

With the popularity of WWW servers, many organizations are trying to build servers that can handle the increased data traffic. Currently, most of these servers are big traditional file servers. Many sites employ distributed file systems across multiple server machines and distribute the requests across these machines to support the request load [1]. These servers typically employ large amounts of memory as cache space for supporting the high request rates on such systems. It is essential that the available memory be used as efficiently as possible to reduce the traffic to disks and also to improve the response time to the user. In this paper, we study several policies for improving the performance of caching in such systems.

Studies on earlier file system request behavior [2] have motivated the design of later file systems [3], specially the caching algorithms and data organization on disks. Most of the earlier studies included none or very few applications that retrieved video or audio data. New forms of data such as audio and video are being used more widely now. Also, the current internet servers support wide use of images in the form of icons. It is not clear if the traditional caching policies would work as well in the new systems where images, audio and video accesses are significant. In this paper, we study cache organizations of a multimedia server that supports text, images, audio and video.

For scalability reasons and for cost-effectiveness, many organizations are building clustered systems to serve the internet traffic [1]. A clustered server organization is shown in Fig. 1. The server has two sets of nodes, *storage nodes* and *network nodes*. Magnetic disks are attached to the storage nodes and users are connected to the server through network nodes. Such organizations are employed at NCSA [1], in many videoservers, for example [4, 5]. The separation of functions into network and storage nodes, either physically or logically, has a number of benefits: (i) easier management of storage and network bandwidth requirements and (ii) easier adaptability of the system to different network interfaces.

Network nodes and storage nodes both may employ caches for storing recently accessed blocks. If the functions of a network node and a storage node are physically separated, data may be cached at a storage node and at several network nodes at the same time. Caches are normally managed locally at a node. If a requested block is missing at the network node cache where the request originates, that request is routed to the storage node of the requested block. What is the best way to manage the different caches in such a server? How well recently proposed cooperative caching [6, 7] work in such a system?

We will use traces obtained from NCSA's web server to evaluate several cache replacement policies and show that performance can be improved compared to LRU policy. We also evaluate the cache sharing approaches for multimedia workloads and show that these can improve performance significantly in a clustered web server.

The rest of the paper is organized as follows. Section 2 discusses the various caching algorithms considered and how they were implemented in the

*This work is supported in part by an NSF Career Award and by a grant from State of Texas Higher Education Board.

1094-7256/97 $10.00 © 1997 IEEE

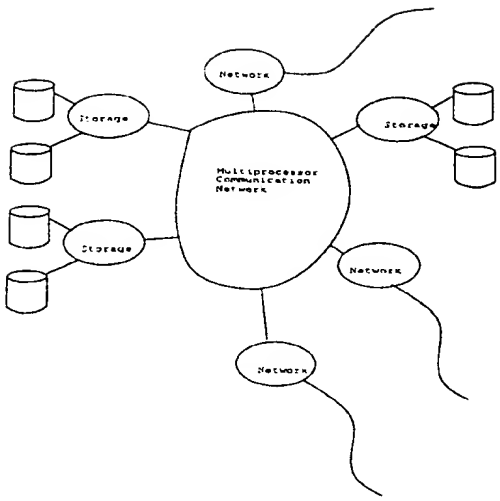

Fig. 1. System model of a clustered server.

simulator. Section 3 discusses the traces used in this study and describes the simulation experiments. Section 4 presents a discussion of the results and Section 5 concludes the paper with directions for future research.

2 Caching algorithms

LRU is widely used as a replacement policy. It is well understood and shown to perform well in a wide variety of workloads. LRU replaces the least recently used block in the cache to make room for a new block. In this section, we consider algorithms that take size of the request and the data type into account. These algorithms may be able to exploit the multiple data types and different sizes of requests to improve the request response time in a multimedia server.

In the first algorithm we consider, text and image data are managed by an LRU policy and audio and video are always left at the LRU end of an LRU chain, i.e., audio and video are managed by an MRU policy. Audio and Video data tend to be large objects and by using an MRU policy to handle them, we expect to limit the impact on other cache contents. Using MRU policy to handle audio and video is equivalent to stealing some of the cache space for buffering the data when audio or video data is accessed. This buffer space is returned to the cache if the audio/video data is not immediately accessed again. We chose to implement an MRU policy for audio/video data so as to dynamically change the amount of buffer space that can be allocated to such data. Such an implementation also enables a fair comparison with other policies using the same amount of cache memory. We call this policy AVI-MRU.

The second algorithm we consider is based on the size of the request. Any request larger than a certain preset threshold (32 KBytes in our simulations) is managed by an MRU policy. Requests smaller than this threshold are managed by an LRU policy. All the requests share the same LRU chain. But blocks of the larger requests, after an access, are placed at the LRU end of the LRU chain. Again, it is expected that this policy would limit larger requests from evicting too many smaller requests from the cache. We call this policy SIZE-MRU.

The third policy we consider is a variation of SpacexAge algorithm. SpacexAge algorithm is shown to be quite effective in minimizing misses in a mass storage system [8, 9]. In these systems, the product of size and age (since last use) of the file is used in determining the candidates for migration to tertiary storage. The variation we considered maintains a number of LRU chains instead of the usual single chain. The chains are partitioned based on request size. In our implementation, requests with sizes up to 2 blocks were put on chain 1, requests between 3 and 8 blocks on chain 2, requests between 9 and 16 blocks on chain 3, and requests above 16 blocks were put on chain 4. A block is 4 KB. Each of these chains are managed by an LRU policy. When a block needs to be replaced from the cache, the space × age product of the four LRU blocks (on the four different chains) are evaluated and the block with the largest space × age product is replaced. It is to be noted that the cache space is not partitioned based on sizes and the cache space can be used by requests of any size. However, larger requests tend to occupy cache space for less time than smaller requests. We call this policy SpacexAge.

Least Frequently Used (LFU) is recently shown to improve cache performance in web servers [10]. LFU keeps track of the number of times a block is accessed and replaces the least frequently used block to make room for an incoming block. It was observed by [10] that a number of blocks/objects are accessed only once and this information was used to modify LFU to LFU*-aging policy. LFU*-aging allows an incoming block to replace only blocks with a frequency count of 1. To make blocks available for replacement, it uses an aging policy where frequency count of the blocks is decremented at regular intervals. Performance results in [10] suggest that LFU*-aging performs better than any other policy they considered.

LFU is more complex to implement than LRU. In LRU, a referenced block is just placed at the head of the LRU chain as the MRU block. In LFU, the referenced block needs to be placed in a sorted (by frequency of use) list and this in general is more complex. LFU*-aging increases the complexity by requiring a timer based aging (reduction of frequency of use) of blocks. Even though this policy is more complex to implement, we consider it so as to compare it with other policies.

We also evaluated a cooperative caching policy called greedy forwarding. In this policy, if a request misses at a network node cache in a clustered web

server, it is forwarded to the storage node cache. A document is cached at only one storage node and the node id is determined by hashing the document id to the number of storage nodes in the system. If the request results in a miss at the storage node as well, it is forwarded to one of the network node caches if it is cached at any other network node. That network node cache then satisfies the request and the requested block will be cached at the originating node. If the requested block is not present at any network node, then it is accessed from the disk. In greedy forwarding, caches at each node are independently managed but enough information is maintained to access data from another node cache. With high-performance networks, data is more efficiently accessed over the network from another node's cache than the disk at the storage node.

In this paper, we evaluate the following caching policies: (i) different replacement policies (compared to LRU), (ii) cooperative sharing in a clustered system.

3 Simulations

Trace driven simulation is used for measuring the effectiveness of various caching policies. We obtained traces from NCSA's WWW server. NCSA records every access made to its web server using a httpd daemon [1]. For details about the trace collection, please see [1]. We obtained traces of the NCSA server for the week of 17th June to 21st June 1996. Each record in this trace contains: the name of the accessed object/file, time of access, the size of the object along with other information. From the description of the objects, we can deduce information about the type of data being accessed whether it is text, images, audio or video. Occasionally, the WWW server may experience errors and these were thrown out of the trace. Similarly, aborted accesses to files were dropped from the trace. We used traces of three days of 18th, 19th and 20th of June as input to our simulator. Each record in our final trace has the following information: day and time of access, name/id of the object, size of the object, type of the object and the origin of the access. The origin of the access is the id of the network node where the request is received. The traces contain accesses to files and not individual blocks within a file. It is assumed that all the blocks within a file are accessed. All requests are read accesses.

The traces of the three days were combined into one long trace. The final trace had 936,170 requests for a total of 2.99 million block (4KB blocks) accesses. The statistics about different accesses are given in Table 2. Based on the number of requests, the current usage of audio and video over the internet is still minimal. However, these data types on an average access many more blocks per request than text or images. It is noted that there are more requests for images than for text, but the images access fewer data blocks. Frequent use of small icons on web pages is the main reason for this behavior. About 170,000 requests from the trace were used to warm up the caches and the rest of the trace was used to gather performance statistics.

We use block hit ratio, request hit ratio and request service time as measures of performance. Current file systems treat all the blocks equally and only look at block hit ratio. However, users are concerned with requests and not individual blocks in the requests. Moreover, misses are not all equal at the disk. Disks are more efficient in serving larger requests than serving many smaller requests that fetch the same amount of data from the disk. Hence, we incorporate this measure of disk service time into our cache model by considering the request service time. Request service time is the amount of time taken to serve a request, with cache hits taking zero time and cache misses taking appropriate amount of time at the disk. Message transfer across the cluster network is assumed to take 650 us based on the performance of current ATM switches [6]. It is assumed that the file layout on the disk is such that 4 blocks of data can be contiguously read on an average on a request miss. In our tests on one of our machines, we found that on an average, it is possible to read up to 8 blocks of data in one disk access. The assumption of 4 blocks is a conservative assumption and the results are in favor of LRU and LFU*-Age compared to SpacexAge.

4 Results

Fig. 2. shows the comparison of various caching algorithms as the memory at the server is varied from 1MB to 256MB. LFU and LFU*-aging perform better than the other policies when the server memory is small, in the range of 1-16MB. However, it is reasonable to expect that an internet server machine will be configured with greater than 16MB of memory. When more memory is available, in the range of 16MB-256MB, LFU and LFU*-Age do not perform significantly better than LRU. SIZ_MRU exhibits similar behavior, performing better than LRU at smaller cache sizes and not doing so well at higher cache sizes. AVI_MRU and SpacexAge policy perform better than LRU in the entire range of 1MB-256MB.

The request hit rate is shown in Fig. 3. A request is said to miss even if one of the blocks in the request needs to be fetched from the disk. It is observed that AVI_MRU doesn't perform as well when request hit rate is compared. Again, LFU and LFU*-AGE perform significantly better than LRU in the range of 1-16MB. SpacexAge achieves consistently better request hit rate than LRU in the entire range of 1-256MB.

Based on the results in Fig.2 and 3, we will limit our further attention to LFU*-AGE, SpacexAge and LRU policies to avoid data clutter. Fig. 4 shows the average request service time with these three policies as the cache size is varied from 1MB-256MB.

SpacexAge consistently outperforms LRU when request service time is considered. Again LFU*-Age performs better than LRU in the range of 1-16MB and doesn't perform as well when there is larger

Table 2. Statistics about file accesses.

Type of request	requests		blocks		Avg. Req. Size	# of distinct
	#	%	#	%	in blocks	files accessed
Text	390,329	41.70	1,645,016	55.02	4.21	14,361
Images	544,389	58.15	1,247,139	41.71	2.29	8,137
Audio	703	0.07	8,939	0.29	12.72	106
Video	749	0.08	89,040	2.98	118.88	90
Total	936,170	100.00	2,990,134	100.00	3.19	22,694

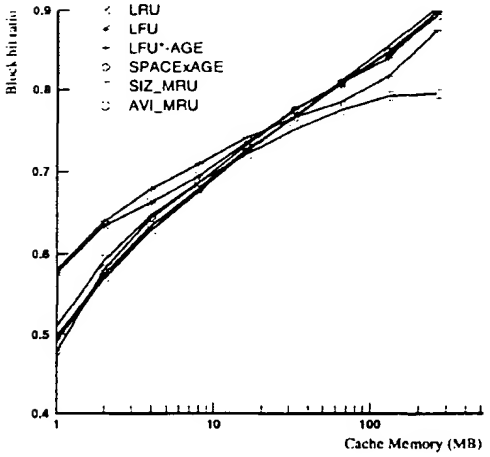

Fig. 2. Block hit ratio.

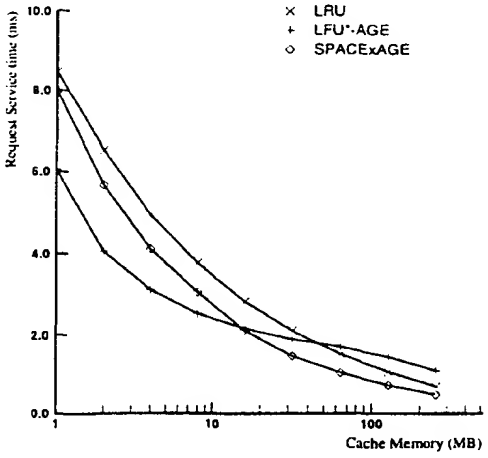

Fig. 4. Request service time.

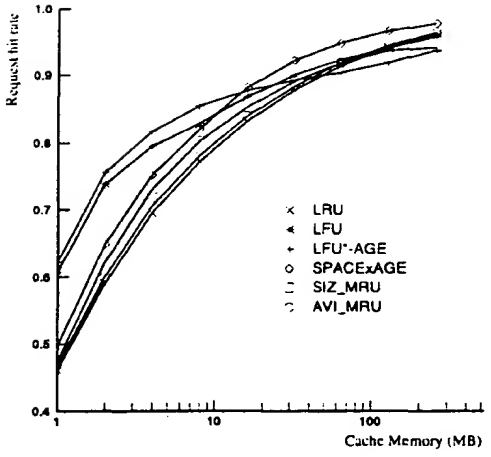

Fig. 3. Request hit ratio.

amount of cache space available. We will consider SpacexAge policy only further because of its consistent performance.

As explained earlier, SpacexAge favors smaller requests over larger requests by evicting the larger requests more quickly from the cache. It is necessary then to examine the extent to which larger requests get penalized as a result. Fig. 5. shows the request service time by object type with LRU and SpacexAge policies. It is observed that SpacexAge provides significantly better service times for text and images than it loses on the larger audio and video requests. Larger requests can be more efficiently serviced at disk than many smaller requests that fetch same amount of data from the disk. SpacexAge policy exploits this property of disks in obtaining better service times for smaller requests (text and images) by favoring them to stay longer in the cache compared to larger requests (Audio and Video).

Fig. 6. shows the request response time with and without cooperative caching. It is observed that cooperative caching improves request response time considerably independent of the replacement policy.

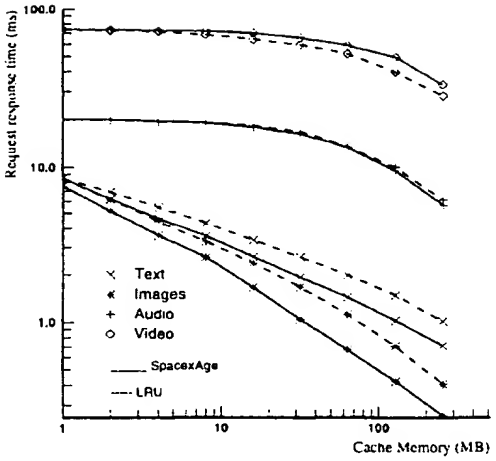

Fig. 5. Service time by object type.

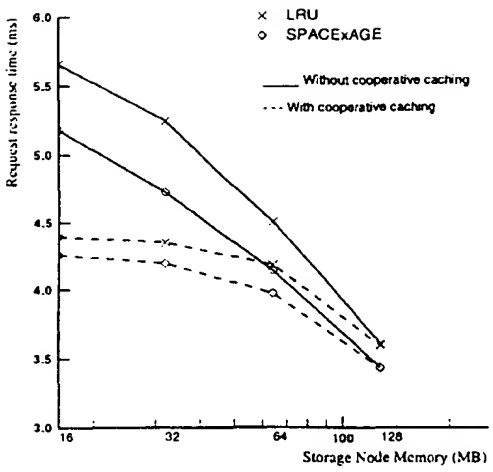

Fig. 6. Request response time with and without cooperative caching.

It is also noted that SpacexAge algorithm continues to have better request response times than LRU even when cooperative caching is employed. As we increase the storage node memory from 16MB to 128 MB, keeping the network node memory fixed at 16MB, the impact of cooperative caching on request response time diminishes. A relatively larger storage node cache results in fewer chances for finding a block at another network node on a miss at the storage node. This is the reason for the diminishing returns. Cooperative sharing improves request response times by up to 35%.

Fig. 7. shows the impact of size of the network node cache on performance. For this set of experiments, the storage node memory is fixed at 128 MB. Larger the network node cache, higher the hit ratio at the network node for an arriving request. When cooperative caching is employed, the size of the network node cache improves performance more significantly. A larger network node cache implies higher chances of finding a block that misses at the storage node cache. SpacexAge continues to have better request response times than LRU. But the relative contribution of replacement policy diminishes as the size of the network node cache is increased.

From figures 4 and 6, it is observed that both cache replacement policy and cooperative caching can have a significant impact on performance (whether or not they are implemented together). SpacexAge is shown to consistently provide better request service times than LRU. Greedy forwarding is shown to significantly improve performance in a clustered web server.

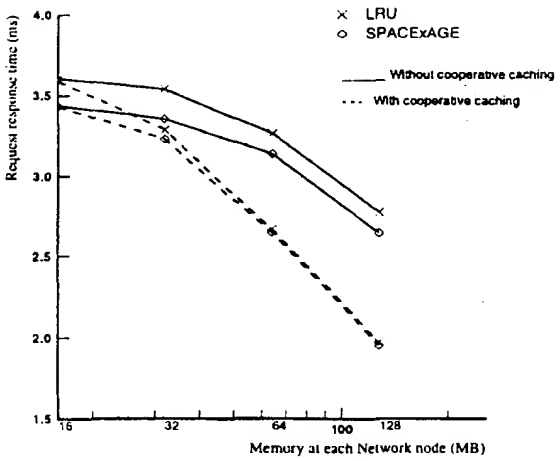

Fig. 7. Request response time as a function of network node cache size.

To study the impact of increased video usage, we created video-enhanced workloads by modifying the traces used earlier. Some of the files in the trace were randomly marked as video files at the beginning of the simulation. Correspondingly, we increased the size of these requests to match the average size of a video request (120 blocks). By doing so, we have not altered the request behavior (order of arrival and frequency of access) but only the size (and type) of the requests. These results are expected to give an indication of how well the various policies perform as the video usage increases.

Results based on these video-enhanced workloads are presented in Fig. 8. We calculated the relative

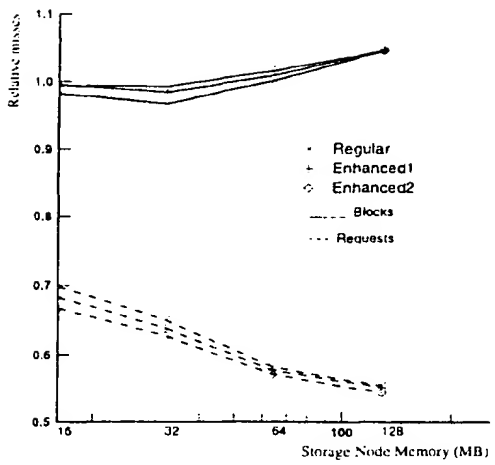

Fig. 8. Relative block and request misses.

(block and request) misses of SpacexAge policy compared to LRU policy. A request is considered to miss if any block of that request requires a disk access. The relative request misses are below 70% for all the workloads. Relative request miss rate decreases as we increase the number of video requests in the workload. Larger video requests evict many smaller requests from the cache. In LRU, these larger requests stay in the cache longer and hence impact the requests over a longer period of time. SpacexAge, by reducing the time larger requests spend in the cache, improves the cache utilization by the smaller requests. The more video requests, the more impact on cache contents and hence the better relative request miss rates of SpacexAge. The relative block miss rate is close to 1.00 (i.e., nearly same number of blocks are missed in both cases) and can be slightly higher than 1.0. The relative block miss rate however increases as the workload has more video requests. This is because of the fact that on an average the size of a missed request is larger in SpacexAge compared to LRU. This clearly shows that SpacexAge policy decreases the number of request misses and when coupled with an efficient disk allocation policy can deliver better response times.

5 Conclusions and Future Work

We presented an evaluation of cache replacement policies that take account of the size of the request or the type of data being accessed. We showed that among the policies considered, SpacexAge algorithm performed the best. We showed that SpacexAge can have better request response times than LRU. SpacexAge is shown to minimize the number of request misses considerably. We also showed that as the video usage grows, SpacexAge does better at minimizing the number of request misses. With efficient disk allocation policies, this decreased request miss rate (with possibly higher block request rate) can translate into better response times.

We also presented a comparison of LRU and SpacexAge with or without cooperative sharing. Cooperative sharing is shown to improve performance significantly for all replacement policies. We also showed that SpacexAge continues to perform better than LRU in all these cases.

In this study, we used the same cache management algorithms both at the storage nodes and network nodes. We plan on evaluating the impact of employing different cache management algorithms at storage nodes and network nodes and how well they work together with the replacement policies studied here.

Web access trends are continuously changing as new applications are developed. It is likely that there is more audio and video access across the web than an year earlier. It is also expected that new streaming protocols for multimedia delivery will alter the server load characteristics. Hence, this work may have to be repeated at frequent intervals to reestablish the validity of the results. We are in the process of validating these results with the workloads at other web servers.

References

[1] T. Kwan, R. McGrath, and D. Reed. NCSA's world wide web server: Design and performance. *IEEE Computer*, pages 68–74, Nov. 1995.

[2] J. K. Ousterhout, H. DaCosta, D. Harrison, J. Kunze, M. Kupfer, and J. Thompson. A trace-driven analysis of the unix 4.2 bsd file system. *Proc. 10th Symp. on Operating System Principles*, pages 15–24, Dec. 1985.

[3] M. McKusick, W. Joy, S. Leffler, and R. Fabry. A fast file system for UNIX. *ACM Trans. on Comp. Systems*, pages 181–197, Aug. 1984.

[4] R. Haskin. The shark continuous-media file server. *Proc. of IEEE COMPCON*, Feb. 1993.

[5] Microsoft. The tiger video server. *Microsoft Press Release*, Apr. 1994.

[6] M. Dahlin, R. Wang, T. Anderson, and D. Patterson. Cooperative caching: Using remote client memory to improve file system performance. *Proc. of 1st Symp. on Oper. Sys. Design and Implementation*, pages 267–280, Nov. 1994.

[7] A. Leff, P. Yu, and J. Wolf. Policies for efficient memory utilization in a remote caching architecture. *Proc. of 1st Int. Conf. on Par. and Dist. Info. Sys.*, pages 198–207, Dec. 1991.

[8] D. H. Lawrie, J. M. Randal, and R. R. Barton. Experiments with automatic file migration. *IEEE Computer magazine*, pages 45–55, July 1982.

[9] A. J. Smith. Long term file migration: development and evaluation of algorithms. *Comm. of ACM*, pages 521–532, Aug. 1981.

[10] M. F. Arlitt and C. L. Williamson. Trace-driven simulation of document caching strategies for internet web servers. *Tech. Report, Univ. of Saskatchewan, Canada*, Sept. 1996.